01/23/25

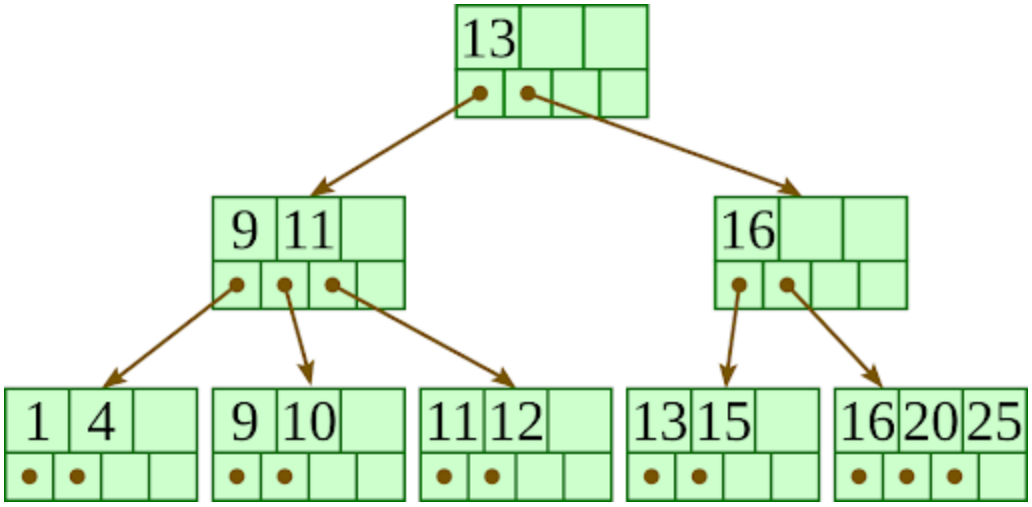
B+ Trees

Optimized for disk-based indexing

Minimize disk accesses for indexing

An m-way tree with order M

- M: maximum number of keys in each node
- M + 1: maximum number of children of each node



M = 3

Children to the left of a node are less than that node

Children to the right of a node are greater than or equal to that node

Properties

- All nodes (except the root) must be half full (minimally)
  - Based on children, not number of keys
- The root nodes doesn't have to be half full
- Insertions are always done at the leaf level
- Leaves are stored as a doubly-linked list

- Keys in nodes are kept sorted

Insertion

- Use keys (pointers) in internal nodes to find correct leaf node
- If the leaf has space, then insert there
    - Can move around the keys in that node to keep numerical order
- If the leaf node is full, then create a new node to split the original node in two
    - Lower half goes into the left (original) node, upper half goes into the right (new) node
    - Includes the value that is being inserted
- After splitting the node into two, create a new parent node and copy the smallest value in the new (right) node to that node
    - The tree only gets deeper (another level) when the root node splits and a new root node is created
    - Splitting internal node → smallest value in the new (right) node gets moved to the parent node
    - Splitting leaf node → smallest value in the new (right) node gets copies to the parent node

Internal nodes: only store keys and pointers (to children)

- Acts as an index
- In B-trees (which are for in-memory indexing), internal nodes store keys and data

Leaf nodes: store keys and data

Searching

- Use the keys in the internal nodes as pointers to determine which way to go
- The data that is being searched for will be found in the leaf node
- Since the leaf nodes are stored as doubly-linked lists, you can do range select